UNISYS

OS 2200
Expipe
User Guide

Release SB6 November 1995

Printed in U S America
Priced Item 7846 7024–000

Contents

Figures

About This Guide

Purpose

This guide describes Expipe, which provides an inter-run first-in first-out (FIFO) file capability for application programs developed using ASCII COBOL and UCS COBOL as well as the Sort/Merge processor (@SORT).

Scope

This guide introduces you to Expipe and provides detailed information on how to use Expipe with your application programs.

Audience

This guide is intended for systems and applications programmers and analysts who are already familiar with the COBOL programming language.

Prerequisites

To use this guide effectively, you should have a thorough understanding of specific application programs at your site as well as familiarity with the COBOL programming language, @SORT processor, and Exec Control Language (ECL) concepts.

How to Use This Guide

Read Section 1 through Section 5 for a general description of Expipe and how it is used. These sections are for applications programmers.

Read Section 6 and Section 7 for information on installing and configuring Expipe. These sections are for system administrators.

Refer to Section 8 for a list of Expipe diagnostic messages and their meanings.

Organization

This guide is organized as follows:

Section 1. Introduction

This section introduces Expipe and describes benefits associated with its use.

Section 2. Getting Started

This section describes the COBOL program and @SORT processor used with Expipe. An example runstream is included.

Section 3. Using Expipe Utility Processors

This section provides detailed information on the utility processors used with Expipe. It includes the format of each processor call statement followed by an example with explanation.

Section 4. Operating Characteristics

This section describes Expipe operating characteristics, which are permanent and are included to promote the effective use of Expipe.

Section 5. Operational Considerations

This section describes Expipe operational considerations.

Section 6. Inside Expipe

This section explains the internal structure of Expipe.

Section 7. Expipe Installation and Configuration

This section describes how to install Expipe in your system, including the installation verification procedure. It also describes how to change Expipe system configuration parameters.

Section 8. Diagnostic Messages

This section lists and explains diagnostic messages issued by Expipe and its utility processors.

Related Product Information

The following documents may be helpful to you when using Expipe. Use the version of these documents that corresponds to the level of software in use at your site.

***OS 1100 ASCII COBOL Programming Reference Manual* (UP-8585)**

This manual describes the components, syntax specifications, and structure requirements of ASCII COBOL programs. It is intended for programmers using ASCII COBOL under control of the Exec operating system.

***OS 2200 Universal Compiling System (UCS) COBOL Programming Reference Manual Volume 1: COBOL Statements* (7831 0448)**

This manual documents the components, syntax specifications, and structure requirements of UCS COBOL programs. UCS COBOL generates code for execution in extended mode on 2200 Series systems.

***OS 2200 Universal Compiling System (UCS) COBOL Programming Reference Manual Volume 2: Compiler and System Interface* (7831 0455)**

This manual describes the UCS implementation of the COBOL language. It describes how to compile, link, and debug programs and how to interact with other system software, such as relational databases or Executive requests.

***OS 2200 Sort/Merge Programming Guide* (7831 0687)**

This describes in detail how to use the Sort/Merge software package, which performs sort/merge operations on your application programs.

Notation Conventions

Syntax formats in this guide use

- Italics to denote variable information that you supply
- Brackets [ ] to denote optional information

Reader Response

If you have any questions or comments about this document, you can either mail or fax the Business Reply Mail Form at the back of this document, or send comments to the following Internet E-mail address:

```
rsvldoc@rsvl.unisys.com
```

You can help us by including the following information:

- Your name
- Your company's name (optional)
- Your Internet E-mail address
- The document title and the 11-digit part number for the document (if applicable)
- The SB level or software level (software documents only)
- The system number and plateau level (hardware documents only)
- Any additional information that will help us in responding to your comments

Section 1
Introduction

This section describes Expipe and how it can benefit your programming environment.

1.1. What Is Expipe?

Expipe is a software program that provides an inter–run FIFO (first–in, first–out) file capability for application programs developed in UCS COBOL or ASCII COBOL, and which use the SORT processor (@SORT). As a batch run writes records to a FIFO file, Expipe transfers them to up to three other concurrently executing batch runs, allowing the runs to read the records from the FIFO file in the order in which they were written. The FIFO file is considered a "pipe," which is where the name Expipe comes from.

1.2. Benefits of Expipe

The primary benefit of Expipe is the accelerated execution of a sequence of batch runs. This is because of the FIFO file implementation, which has the following characteristics:

- For each FIFO file, Expipe allocates one shared (APPLICATION-level) bank that is divided into buffers to store blocks of records.
- Expipe does not assign mass storage back–up for FIFO files.
- Each FIFO file is transferred between runs using the buffers without the need for I/O to mass storage.
- Since access to the FIFO file is sequential, as soon as all the succeeding runs finish reading a block in a buffer, Expipe discards the block and prepares the buffer for reuse by another block.

Other benefits of Expipe include

- Faster execution by eliminating I/O to mass storage
- Avoidance of errors due to insufficient mass storage space

For example, suppose there are two batch runs (A and B) that are connected by an intermediate sequential file. They must run in sequence as shown in Figure 1–1. In this situation, run B must wait for run A to finish creating the sequential file.

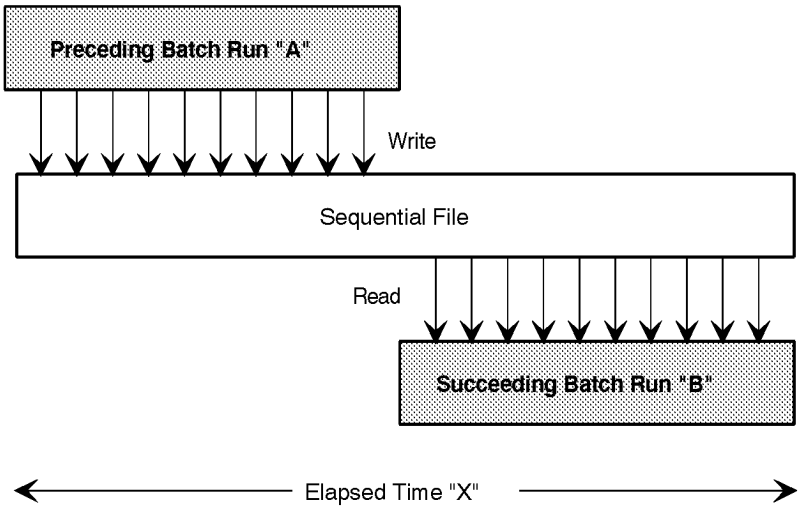


Figure 1–1. Two Batch Runs Using a Sequential File

By replacing the sequential file with a FIFO file as shown in Figure 1–2, the two runs can be executed in parallel. This is because Expipe transfers records from A to B as they are created.

In actual implementation, Expipe synchronizes runs A and B at each block of records. That is, when run A has completed writing a block of records to the FIFO file, then run B can immediately read the block from the FIFO file and start processing each record of the block.

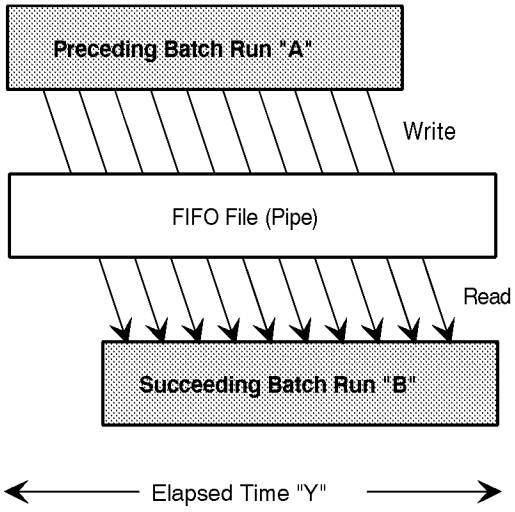


Figure 1–2. Two Batch Runs Using a FIFO File

As shown by these figures, Expipe reduces the elapsed time by up to 30 or 40 percent. In equation form, the time savings are (X–Y)/X = 0.3 to 0.4.

Section 2
Getting Started

This section describes how to use Expipe with a COBOL program and the SORT processor (@SORT). It includes an example Expipe runstream.

2.1. Using Expipe

Expipe can be used from the application programs compiled by UCS COBOL and ASCII COBOL as well as the SORT processor (@SORT). No changes are required to your COBOL programs or SORT processor parameters if they use PCIOS sequential SDF files. Otherwise, some changes are required.

The files defined as follows (implying PCIOS sequential SDF files) can be designated as FIFO files with no changes to your existing application programs:

- For application programs developed in UCS COBOL and ASCII COBOL:

```
SELECT file-name ASSIGN TO DISC
       ORGANIZATION IS SEQUENTIAL
SELECT file-name ASSIGN TO TAPE
       (without "RECORDING MODE F/U/V" for UCS COBOL)
```

- For the SORT processor:

```
FILEIN=name MODE=SDF
FILEOUT=name MODE=SDF
```

A small change is required for your existing runstreams in order to use the FIFO files. This change involves replacing the @ASG,A control statements for PCIOS sequential SDF files with @EXPASG processor call statements that declare use of the FIFO files. In addition, the FIFO files must be defined by the @EXPDEF processor call statements prior to issuing an @EXPASG processor call.

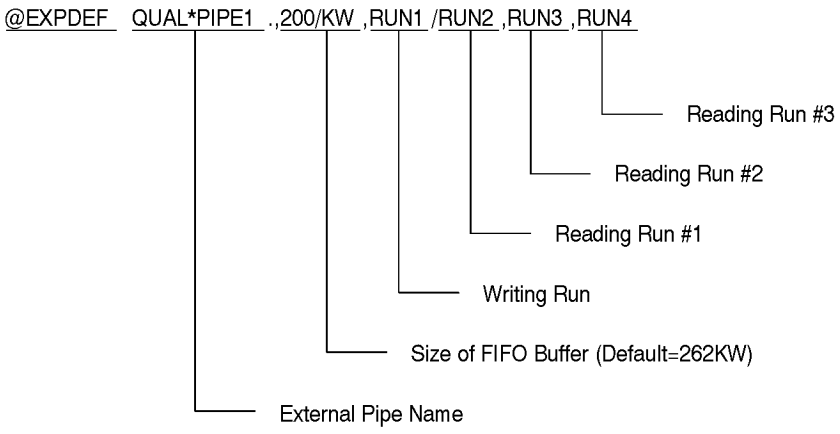
2.2. Example of Using Expipe

When using Expipe, perform the following steps:

1. Define a pipe by @EXPDEF processor call statement.
2. Prepare the runstreams to use the pipe, including @EXPASG processor call statements to assign the pipe in each runstream:
   - Only one runstream is used to write records to the pipe
   - Up to three runstreams can be used to read records from the pipe

The @EXPDEF processor call statement is similar to @CAT control statement; whereas, the @EXPASG processor call statement is similar to the @ASG control statement. See 3.1 for additional information on EXPDEF and 3.2 for information on EXPASG.

The following example shows a pipe definition using an @EXPDEF processor call statement.



The following is an example runstream used to write records to a pipe by a COBOL program:

```
@RUN     RUN1
@EXPASG  QUAL*PIPE1.
@USE     OUT-FILE.,QUAL*PIPE1.
@XQT     PROGRAM
```

The following is an example runstream used to read records from a pipe by the SORT processor:

```
@RUN     RUN2
@EXPASG  QUAL*PIPE1.
@USE     IN-FILE.,QUAL*PIPE1.
@SORT
FILEIN=IN-FILE  MODE=SDF
RECORD=10
```

Pipes are identified by external pipe names, which have a format similar to external file names (for example, qualifier*pipe_name). The @USE and @QUAL control statements can be used for pipes in the same manner as for external file names. When a qualifier is not specified for a pipe name, the run's project–id is used as the qualifier.

Figure 2–1 shows an example using two pipes. The run named SETUP defines two pipes (Q*AA and Q*BB) as follows:

- Pipe Q*AA has a writing run (RUN1) and two reading runs (RUN2 and RUN3). The size of the pipe, or its FIFO buffer, is 262KW (the default).
- Pipe Q*BB has a writing run (RUN2) and a reading run (RUN4). The size of the pipe is 100 blocks.

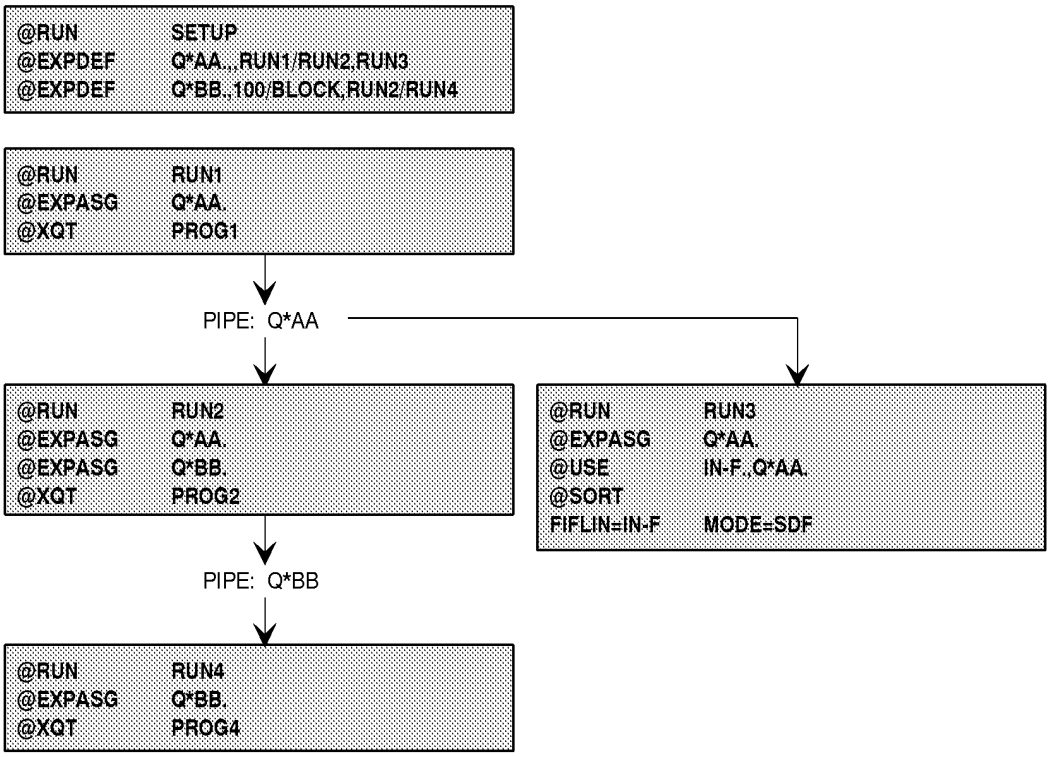


Figure 2–1. Example of Using Expipe

2.3. Using Expipe from COBOL Programs

To use Expipe from COBOL programs, you must specify the following for the file to be connected to a pipe:

- DISC or TAPE as implementor name of SELECT clause
- ORGANIZATION IS SEQUENTIAL
- No RECORDING MODE F/U/V clause for the files with implementor name TAPE in UCS COBOL programs

A pipe can be seen as a COBOL sequential file; however, it will have functional restrictions as discussed in 2.3.2.

The following examples of COBOL SELECT clauses can use Expipe:

```
SELECT file-name  ASSIGN TO DISC [pipe-name]
SELECT file-name  ASSIGN TO TAPE [pipe-name]
```

The following examples of COBOL SELECT clauses cannot use Expipe:

```
SELECT file-name  ASSIGN TO MASS-STORAGE  [pipe-name]
SELECT file-name  ASSIGN TO UNISERVO      [pipe-name]
SELECT file-name  ASSIGN TO PRINTER       [pipe-name]
SELECT file-name  ASSIGN TO DISC          [pipe-name]
                  ORGANIZATION  IS  RELATIVE
SELECT file-name  ASSIGN TO DISC          [pipe-name]
                  ORGANIZATION  IS  INDEXED
```

Notes:

- *Expipe can be used for the files specified as RECORDING MODE SDF in ASCII COBOL programs; however, this is not recommended. It is allowed only because it is impossible to diagnose such usage.*
- *Expipe can be used for the files specified as ASSIGN TO TAPE in the SELECT clause only if the program is designed with the assumption that the actual files are assigned to the disk device with the @ASG control statement.*

2.3.1. Using Multiple Pipes between COBOL Programs

When using multiple pipes between COBOL programs, you must ensure that the writing program and reading program or programs have similar cycles when accessing pipes. The following example shows a writing program and a reading program, each with a different cycle, trying to access two pipes.

Writing Program (WRPROG)

```
      OPEN OUTPUT PIPE1 PIPE2.
      PERFORM WRIT1 nn TIMES.
      CLOSE   PIPE1 PIPE2.
WRIT1.
      WRITE   REC1.           <----(1)
      WRITE   REC2.           <----(2)
```

Reading Program (RPROG)

```
       OPEN INPUT PIPE1 PIPE2.
       PERFORM READ1 UNTIL cond_1.
              :
       PERFORM READ2 UNTIL cond_2.
              :
READ1.
      READ PIPE1 AT END para-1.    <----(3)
READ2.
      READ PIPE2 AT END para-2.    <----(4)
```

In this example, the reading program RPROG has a cycle to access the pipe PIPE2, which is quite different from that of the writing program WPROG. That is, RPROG starts reading PIPE2 after it finishes reading PIPE1; whereas, WPROG writes to PIPE1 and PIPE2 concurrently. Therefore, it is possible for the FIFO buffer for PIPE2 to become full before control is passed to the READ statement (4), which results in both WPROG and RPROG becoming deadlocked.

In order to avoid a deadlock situation, the writing program (WPROG) must be modified so that it can have the cycle to access PIPE1 and PIPE2 similar to that of the reading program (RPROG). The modified writing program is shown in the following example.

Modified Writing Program (WPROG)

```
      OPEN OUTPUT PIPE1 PIPE2.
      PERFORM WRIT1 nn TIMES.
      CLOSE   PIPE1.
      PERFORM WRIT2 nn TIMES.
      CLOSE   PIPE2.
WRIT1.
      WRITE   REC1.               <----(1)
WRIT2.
      WRITE   REC2.               <----(2)
```

2.3.2. Unavailable COBOL Sequential I-O Features

The following features of COBOL Sequential I-O cannot be used with Expipe:

- PADDING CHARACTER clause
- RECORD DELIMITER clause
- RERUN clause
- MULTIPLE FILE TAPE clause
- OPEN I-O statement
- OPEN EXTEND statement
- OPEN REVERSED statement
- OPEN NO REWIND statement
- REWRITE statement
- Implied CLOSE in STOP RUN statement (UCS COBOL)

All of the remaining features of COBOL Sequential I-O can be used with Expipe, including the following:

- OPTIONAL clause
- SAME RECORD AREA clause
- USE declaratives
- File concatenation

2.4. Using Expipe from the SORT Processor

To use Expipe from the SORT processor, you must specify SDF for the MODE parameter. A pipe then can be seen as a PCIOS SDF sequential file; however, it will have functional restrictions as discussed in 2.4.2.

The following is an example of using Expipe with SDF files:

```
@SORT
FILEOUT=pipe_name MODE=SDF
FILEIN=pipe_name MODE=SDF
DISKS=5,,TRK,50
```

The following is an example of file types that cannot use Expipe:

```
@SORT
FILEOUT=pipe_name MODE=ACOB
FILEIN=pipe_name MODE=MSAM,R
```

2.4.1. Using a Pipe as an Input File to the SORT Processor

When using a pipe as an input file to the SORT processor, it is impossible for the SORT processor to estimate the amount of input data and to assign scratch files automatically. Therefore, you must do either of the following:

- Assign scratch files (XA, XB, XC, etc.) manually prior to calling the SORT processor
- Specify one or more of the following parameters to the SORT processor:
  - RECORD parameter
  - NUMREC parameter
  - VOLUME parameter
  - DISKS parameter (with the file size specified)
  - DRUM parameter (with the file size specified)
  - FAST parameter (with the file size specified)

2.4.2. Unavailable SORT Processor Parameters

The following features of the SORT processor for PCIOS SDF sequential files cannot be used with Expipe:

- File assign capability of FILEOUT parameter for an Expipe output file
- SORT processor determining the size of scratch files based on the input Expipe file size
- CKPT parameter
- MOVE parameter
- REELS parameter

Section 3
Using Expipe Utility Processors

This section describes the utility processors used with Expipe. Each processor call statement includes the format along with a detailed explanation and example.

3.1. @EXPDEF Processor Call Statement

Function

@EXPDEF processor call statement defines or deletes a pipe.

Format

```
@EXPDEF[,option] pipe-name.,[size[/unit]],o-run/i-run1[,i-run2[,i-run3]]
```

Explanation

To define a pipe, specify the following on the @EXPDEF processor call statement:

- Pipe name
- Size of FIFO buffer
- Unit for size of FIFO buffer
- Run-ids that can be connected to the pipe

Defined pipes are registered in Expipe and can be assigned to the runs with the specified run-ids using the @EXPASG processor call statement. If the same pipe name is already defined, the @EXPDEF will replace it with the new specifications.

Following are the explanations of the fields on @EXPDEF processor call statement:

option

is the name of the option. You must specify "D" when deleting a pipe.

pipe-name

is the name of the pipe to be defined or deleted.

size/unit

is the size of FIFO buffer and its unit. The size can be either physical (number of kilowords) or logical (number of blocks). Expipe converts the logical buffer size to a physical size, which must be in the range of 65,535 words to 262,143 words.

The physical buffer size can be specified by *n*/KW, where *n* is a value in the range of 65 to 262.

The logical buffer size can be specified by *n*/BLOCK, where *n* is the number of blocks.

When the size is specified but the unit is omitted, BLOCK is assumed for the unit. When both size and unit are omitted, 262KW is assumed.

o-run

is the run–id associated with the writing (output) side of the pipe.

i-run1, i-run2, i-run3

are the run-ids associated with the reading (input) side of the pipe. Up to three run-ids can be specified. At least one run-id (i-run1) must be specified. Different run-ids must be specified for o_run, i_run1, i_run2, and i_run3; none of them can be the same.

Examples of Usage

The following example defines a pipe named EMPLOY*SORT-OUT with a size of 262K words between a run named SORT1 and another run named RPT1.

```
@EXPDEF EMPLOY*SORT-OUT.,,SORT1/RPT1
```

The following example defines a pipe named TEMPORARY with size of 50K words between a writing run named APPL1 and two reading runs named APPL2 and APPL3. This example will generate an error.

```
@EXPDEF TEMPORARY.,50/KW,APPL1/APPL2,APPL3
```

Listing Examples

```
@EXPDEF EMPLOY*SORT-OUT.,,SORT1/RPT1
EXPDEF xRy (yymmdd hhmm:ss) yymmdd hh:mm:ss
EXPIPE101: Completed

@EXPDEF TEMPORARY.,50/KW,APPL1/APPL2,APPL3
EXPDEF xRy (yymmdd hhmm:ss) yymmdd hh:mm:ss
EXPIPE221(ERROR): Buffer size is out of range.  Should be 65KW through 262KW.
```

3.2. @EXPASG Processor Call Statement

Function

The @EXPASG processor call statement assigns a pipe to the run which executed the @EXPASG statement.

Format

```
@EXPASG  pipe-name.
```

where *pipe-name* is the name of the pipe to be assigned to the run.

The pipe must be defined by the @EXPDEF processor call statement prior to issuing the @EXPASG processor call statement. The run that executes the @EXPASG statement must have its run-id specified by the @EXPDEF statement for the pipe as o-run, i-run1, i-run2, or i-run3. Otherwise, the pipe cannot be assigned to the run by the @EXPASG statement.

If a pipe is used more than once in a runstream, the @EXPASG statement must be specified each time. The pipe is automatically freed when the execution of the program that uses the pipe terminates.

The @EXPASG processor internally assigns a temporary file with the same name as the pipe. You cannot free or erase this file. Therefore, you cannot have a file with the same name as a pipe in a runstream.

Listing Example

```
@EXPASG  OUR*PIPE.
EXPASG xRy (yymmdd hhhh:mm) yymmdd hh:mm:ss
Expipe101: Completed
```

3.3. @EXPSTA Processor Call Statement

Function

The @EXPSTA processor call statement displays a pipe's usage status information.

Format

```
@EXPSTA [,option] [pipe-name.]
```

Explanation

option

is the option to be specified as follows:

A	Display usage status information for active (in use) pipes.
B	Display the information on banks used by Expipe.
C	Display value of BUFNUM and ALLABORT configuration parameters (see 7.3) and the BDI values of FIFO banks.
D	Display detailed information for pipes when option A, F, L, U, W, or *pipe-name* is specified.
F	Display usage status information for buffer waiting pipes. Buffer waiting pipes are pipes whose FIFO buffer cannot be allocated because of the BUFNUM limit.
L	Equivalent to specifying options A, B, S, U, and W.
S	Display usage status information for sleep pipes. Sleep pipes are pipes that are defined but have not been used by any runs yet.
U	Display usage status information for used pipes. Used pipes are pipes that were previously used but are not in use currently.
W	Display usage status information for waiting pipes. Waiting pipes are those pipes whose creation (by writing runs) is delayed until reading runs can complete reading a buffer.

pipe-name

is the name of the pipe whose usage status information is to be displayed. You must specify the *pipe-name* if you want to know the usage status of a specific pipe. When *pipe name* is specified, options A, F, S, U, and W are ignored. When *pipe-name* and option L are specified, it is equivalent to specifying *pipe-name* and options B and C.

Example Listing with Explanation

```
@EXPSTA,BCD  EXSAMPLE*Expipe1.
EXPSTA xRy (yymmdd hhmm:ss) yymmdd hh:mm:ss

Pipes  : DEFINED(  22)   MAX(2000)                                          (1)
         ACTIVE (   1) WAITING(   4) WAITBUF(   1) USED(   4) SLEEP(  12)  (2)
Banks  : 210002(code)   limit=001000-000000175561  size= 63.9 KW           (3)
         210171(work)   limit=000000-000001103000  size=296.4 KW           (4)
         210141(buffer) limit=000000-000000176750  size= 65.0 KW           (5)
Config:  ALLABORT = YES                                                     (6)
         BUFNUM = 25                                                        (7)
FIFO Bank BDI:
         210171 210170 210167 210166 210165 210164 210163 210162 210161 210160
         210157 210156 210155 210154 210153 210152 210151 210150 210147 210146
         210145 210144 210143 210142 210141                                 (8)

EXSAMPLE*EXPIPE1.,65/KW,JOB11/JOB21,JOB22                                   (9)
  Status : ACTIVE                                                          (10)
  Buffer : size=65000,  L,BDI=210141,  limit=000000-176750                 (11)
  Define : JOB03A              yyyy/mm/dd hhmm:ss                          (12)
  Connect: JOB11 (PRG012      ) yyyy/mm/dd hhmm:ss -                       (13)
                 Open           yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss   (14)
                 Write          yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss   (15)
                 Close          yyyy/mm/dd hhmm:ss                        (16)
         : JOB21 (PRG021      ) yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss   (17)
                 Open           yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
                 Write          yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss   (18)
                 Close          yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
         : JOB22 (PROG22      ) yyyy/mm/dd hhmm:ss -
                 Open           yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
                 Write          yyyy/mm/dd hhmm:ss -
                 Close          yyyy/mm/dd hhmm:ss
  ------------------------------------------------------------------------
  PRG012                 Sectors=        9832  Wait=hh:mm.ss Stat=ACTIVE  (19)
         ---> PRG021     Sectors=        9650  Wait=hh:mm.ss Stat=NORMAL  (20)
         ---> PRG022     Sectors=        9200  Wait=hh:mm.ss Stat=ACTIVE
  ------------------------------------------------------------------------
```

Note: *For this example, lines (1) through (5) are displayed by specifying option B, and lines (6) through (8) are displayed by specifying option C. Lines (19) and (20) are shown only for ACTIVE or USED pipes. (They are not shown for WAITING, WAITBUF, or SLEEP pipes.)*

1. DEFINED shows the number of the pipes currently defined by the @EXPDEF processor call statement. MAX shows the maximum number of the pipes that can be defined by @EXPDEF processor call statement.
2. ACTIVE shows the number of pipes that are in use. WAITING shows the number of pipes that are waiting to be created (by writing runs) but are delayed until reading runs can complete reading a buffer. WAITBUF shows the number of pipes whose FIFO buffer is not available. USED shows the number of pipes that were used previously but are not in use currently. SLEEP shows the number of pipes that are defined but have not been used by any runs yet.

3. This line shows the following bank information regarding the code bank of the Expipe fixed-gate shared subsystem:
   - L,BDI value
   - Lower and upper limit of offset
   - Size in kilowords
4. This line shows the bank information for the data bank of Expipe fixed-gate shared subsystem. This is where the system-wide pipe definition and status information is maintained.
5. This line shows bank information for the bank allocated as a FIFO buffer for a pipe.
6. This line shows the value of the ALLABORT configuration parameter.
7. This line shows the value of the BUFNUM configuration parameter.
8. These lines show the BDI values of the FIFO buffer banks.
9. This line shows the external pipe name and definition of a pipe. The format of this line is the same as that of the @EXPDEF processor call statement.
10. This line shows the status of the pipe (ACTIVE, WAITING, WAITBUF, USED, or SLEEP).
11. This line shows the following information for the bank allocated as the pipe's FIFO buffer:
    - Size in kilowords
    - L,BDI value
    - Lower and upper limit of offset (in octal)
12. This line shows the run-id (defined by the @EXPDEF processor call statement) and the date and time the pipe was defined.
13. This line shows the following information:
    - Run-id for the writing side of the pipe
    - Name of the program that writes to the pipe
    - Date and time the writing program started execution
    - Date and time the writing program finished execution
14. This line shows the date and time when Expipe started the opening process and the date and time when Expipe finished the opening process.
15. This line shows the date and time when Expipe wrote a first data block into the pipe and the date and time when Expipe finished writing all data blocks into the pipe.
16. This line shows the date and time when Expipe started the closing process and the date and time when Expipe finished the closing process.

17. This line shows the following information:
    - Run-id for the reading side of the pipe
    - Name of the program that reads from the pipe
    - Date and time the reading program started execution
    - Date and time the reading program finished execution
18. This line shows the date and time Expipe read the first data block from the pipe and the date and time Expipe finished reading data blocks from the pipe.
19. This line shows the following information:
    - Name of the program that writes to the pipe
    - Number of sectors written to the pipe
    - Accumulated amount of time the program has waited due to the FIFO buffer being full
    - Status of the writing program as follows:
      – ACTIVE (program executing)
      – NORMAL (program terminated execution normally)
      – ERROR (program terminated execution in error)
20. This line shows the following information:
    - Name of the program that reads from the pipe
    - Number of sectors read from the pipe
    - Accumulated amount of time the program has waited because the pipe was not created or the FIFO buffer was empty
    - Status of the reading program as follows:
      – ACTIVE (program executing)
      – NORMAL (program terminated execution normally)
      – ERROR (program terminated execution in error)

Example 2

```
@EXPSTA,BAUWS
EXPSTA xRy (yymmdd hhmm:ss) yymmdd hh:mm:ss

Pipes  : DEFINED(   5)   MAX(2000)
         ACTIVE (   1) WAITING(   1) WAITBUF(   1) USED(   1) SLEEP(   1)
Banks  : 210002(code)   limit=001000-000000175561  size= 63.9 KW
         210171(work)   limit=000000-000001103000  size=296.4 KW
         210141(buffer) limit=000000-000000176750  size= 65.0 KW
         210142(buffer) limit=000000-000000377777  size= 128.0 KW

EXSAMPLE*EXPIPE1.,65/KW,JOB11/JOB21,JOB22
  Status : ACTIVE
  Buffer : size=65000,  L,BDI=210141,  limit=000000-176750
  Define : JOB03A              yyyy/mm/dd hhmm:ss
  Connect: JOB11 (PRG012      ) yyyy/mm/dd hhmm:ss -
         : JOB21 (PRG021      ) yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
         : JOB22 (PRG022      ) yyyy/mm/dd hhmm:ss -
  ------------------------------------------------------------------------------
  PRG012                Sectors=        9832  Wait=hh:mm.ss Stat=ACTIVE
         ---> PRG021    Sectors=        9650  Wait=hh:mm.ss Stat=NORMAL
         ---> PRG022    Sectors=        9200  Wait=hh:mm.ss Stat=ACTIVE
  ------------------------------------------------------------------------------

EXPIPE*DEFINE1.,10/BLOCK,JEA30/JEA31
  Status : USED
  Define : DEF-P                yyyy/mm/dd hhmm:ss
  Connect: JEA30 (GYM203-W     ) yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
         : JEA31 (GYM203-R     ) yyyy/mm/dd hhmm:ss - yyyy/mm/dd hhmm:ss
  ------------------------------------------------------------------------------
  GYM203-W              Sectors=         894  Wait=hh:mm.ss Stat=NORMAL
         ---> GYM203-R  Sectors=        9650  Wait=hh:mm.ss Stat=ERROR
  ------------------------------------------------------------------------------

EXPIPEJOB50*EXPIPENAME01.,,JOB51/JOB52
  Status : WAITING   hh:mm.ss
  Buffer : size=0     ,  L,BDI=000000,  limit=000000-000000
  Define : DEF-P                yyyy/mm/dd hhmm:ss
  Connect: JOB51 (             )     /  /       :   -
         : JOB52 (GYM503-R     ) yyyy/mm/dd hhmm:ss -

EXPIPEJOB60*EXPIPENAME01.,200/KW,JOB61/JOB62
  Status : WAITBUF   hh:mm.ss
  Buffer : size=0     ,  L,BDI=000000,  limit=000000-000000
  Define : DEF-P                yyyy/mm/dd hhmm:ss
  Connect: JOB61 (GYM603-W     ) yyyy/mm/dd hhmm:ss -
         : JOB62 (             )     /  /       :   -
```

Section 4
Characteristics of Expipe

This section describes operational considerations that are permanent. These considerations will help to promote the effective use of Expipe.

4.1. Program Execution Modes

While the main target of Expipe is batch programs, it can also be used by demand programs. Expipe cannot be used by the following programs:

- Real-time mode programs
- TIP programs
- HVTIP programs
- Multiactivity programs

4.2. Naming a Pipe

Pipes are identified by the pipe names, which have a format similar to external file names (for example, *qualifier*pipe_name*). The @USE and @QUAL control statements can be used for pipes in the same manner as for external file names. If a qualifier is not specified for a pipe name, the run's project-id is used as the qualifier.

4.3. Upper Limits of Expipe

The upper limits of Expipe are as follows:

- Maximum number of pipes that can be defined in a system is 2000
- Maximum number of pipes that can be concurrently used in a system is 100 (This number can be changed; see 7.3.2)
- Maximum number of runs that can be read from a pipe is 3
- Maximum number of pipes that can be concurrently used in a run is 20
- Maximum size of FIFO buffer for a pipe is 262,143 words

4.4. Waiting for Pipe Availability

After assigning a pipe using the @EXPASG processor control statement, a reading run waits for the writing run to complete execution of the @EXPASG statement. Expipe displays a message on your system console every 30 minutes when the reading run waits for @EXPASG by the writing run for more than 30 minutes.

4.5. Creating and Purging a Pipe

A pipe is created when the writing program opens the pipe for output. The creation of a pipe means that Expipe has allocated a shared (APPLICATION level) bank as a FIFO buffer for the pipe and is ready for data transfer.

If a pipe has not been created when a reading program opens a pipe for input, the reading program waits for the creation of the pipe by a writing program. Writing programs, however, can start writing records to the pipe without waiting for a reading program to open the pipe for input.

A reading program can start reading records from a pipe when it opens the pipe for input and the writing program opens the pipe for output. Expipe displays a message on the system console every five minutes, when a reading program waits for the creation of the pipe by the writing program for more than five minutes.

A pipe is purged when all of the programs that previously opened the pipe have terminated. Purging a pipe means that Expipe has freed the shared bank used as the FIFO buffer for the pipe and has marked the pipe unavailable.

4.6. Writing to and Reading from a Pipe

A block of records written to a pipe is stored in a FIFO buffer. These records are discarded when all of the reading programs have finished reading the FIFO buffer. If the reading programs are significantly slower at accessing the pipe than the writing program, the FIFO buffer may become full, which forces the writing program to wait. If the writing program is significantly slower at accessing the pipe than the reading programs, the FIFO buffer may become empty, which forces the reading program to exit. This means that the writing program and reading programs for a pipe must have similar access cycle times to the pipe.

Expipe displays a message on the system console every five minutes if the pipe is full and the writing program cannot write to the pipe for more than five minutes. Likewise, Expipe displays another message on the system console every five minutes if a pipe is empty and the reading programs cannot read from the pipe for more than five minutes, forcing reading programs to wait.

4.7. Error Detection and Recovery

When a program using a pipe has terminated in error, the data transferred using the pipes cannot be recovered. Therefore, it is necessary to re-execute the runs from a point prior to the start of using the pipes.

When a writing program has terminated in error after opening a pipe for output, all of the reading programs that have opened the pipe for input will terminate in error (after printing a diagnostic message).

When a reading program has terminated in error after opening a pipe for input, the writing programs that have opened the pipe for output will terminate in error unless there is at least one remaining reading program executing normally.

When a writing program has terminated in error before opening a pipe for output, all of the reading programs that have opened the pipe for input will keep waiting for the creation of the pipe by the writing program. Therefore, it is necessary to resolve the cause of the error and then to re-execute the writing program.

When a reading program has terminated in error before opening a pipe for input, the writing program starts writing records to the pipe and eventually stalls as the FIFO buffer becomes full. Therefore, it is necessary to resolve the cause of the error and then to re-execute the reading program.

When an error is detected in Expipe during execution of a user program, the following error processing takes place:

- Expipe either displays an error message or returns an error code to PCIOS. For contingency errors, Expipe produces a dump listing after contingency processing.
- PCIOS returns the error status returned by Expipe to its caller (URTS, ACOB Runtime Library, or @SORT processor).
- URTS, ACOB Runtime Library, or @SORT processor displays an error message indicating an error return from PCIOS and performs standard error processing.
- If a program has terminated without closing a pipe, Expipe purges the pipe.

4.8. When a System Failure Occurs

If a system failure takes place while pipes are being used, the pipes are lost along with the data being transferred. However, pipe definitions and usage status remain intact unless an initial boot (I-boot) is done. Runs lost by a system failure can be reexecuted from a point prior to the start of pipe use.

4.9. No Interhost Pipe Support

Expipe does not support interhost pipes in a multiple-host environment (such as XTC).

Section 5
Operational Considerations

This section describes Expipe permanent operational considerations. These considerations are included in this guide to ensure that you use Expipe effectively.

5.1. Using Pipes with Newly-Developed Programs

When using pipes with newly-developed programs, Unisys recommends that you start using Expipe after making sure that the programs work with disk files.

5.2. Access to Pipes Not through PCIOS C2SSDF

Any attempt to access a pipe other than through PCIOS C2SSDF ends up in I/O to a temporary file with the same name as the pipe, which the @EXPASG processor assigns internally.

5.3. Access to Pipes through PCIOS C2SSDF from Programs other than ACOB, UCOB, or SORT

When attempting to access a pipe through PCIOS C2SSDF from programs other than ASCII COBOL (ACOB), UCS COBOL (UCOB), or the SORT processor, such programs will terminate in error when attempting to open the pipe for input or output. If a user (erroneously) intends to use such programs as writing or reading programs for a pipe coupled with ACOB, UCOB, or SORT, then the program connected to the other end of the pipe will also have errors (as explained in 4.7).

Note: *Expipe cannot detect erroneous access to a pipe from a Query Language Processor (QLP 2200) program or a UCS FORTRAN program. Do not use Expipe with these types of programs.*

5.4. Using Expipe on a Memory-Tight System

On memory-tight C–series systems (such as the 2200/400), it is possible that Expipe will not improve performance. This is especially true when the FIFO buffers in APPLICATION level banks are swapped in and out frequently.

5.5. Applying Expipe Gradually to Batch Runs

When using Expipe with writing runs and reading runs that affect each other (as explained in 4.6 and 4.7), Unisys recommends that you apply Expipe gradually to batch runs by first focusing on I/O bottlenecks of intermediate sequential files.

5.6. Avoiding the X Keyin

You must never use an X keyin to kill a run that is currently using Expipe. The X keyin will cause Expipe to malfunction. This is because the X keyin prevents the purge of a pipe, which includes the freeing of the FIFO buffer and the initialization of a pipe.

5.7. Avoiding the E Keyin (@@X T)

You should avoid using an E keyin from the system console, or an @@X T from a demand terminal, to terminate the Expipe utility processors (@EXPDEF, @EXPASG, and @EXPSTA) when first executing Expipe after a system boot. The E keyin will cause Expipe to malfunction because it prevents the initialization of the Expipe FGSS.

5.8. No Support for Checkpoint/Restart (CKRS)

Expipe does not support Checkpoint/Restart (CKRS). This means that CKRS cannot be used for any run that makes use of Expipe.

5.9. Collection of the ACOB Program

Expipe internally uses a PROGRAM-level working bank whose address range is 030000 through 031777. The data bank of the ACOB user program must be allocated to an address greater than or equal to 032000 or else the program may abort with an IGDM (illegal BDI).

Section 6
Inside Expipe

This section describes the internal structure of Expipe.

Expipe consists of three utility processors (@EXPDEF, @EXPASG, and @EXPSTA), one common bank, and one fixed-gate shared subsystem (FGSS). These components are shown in Figure 6–1.

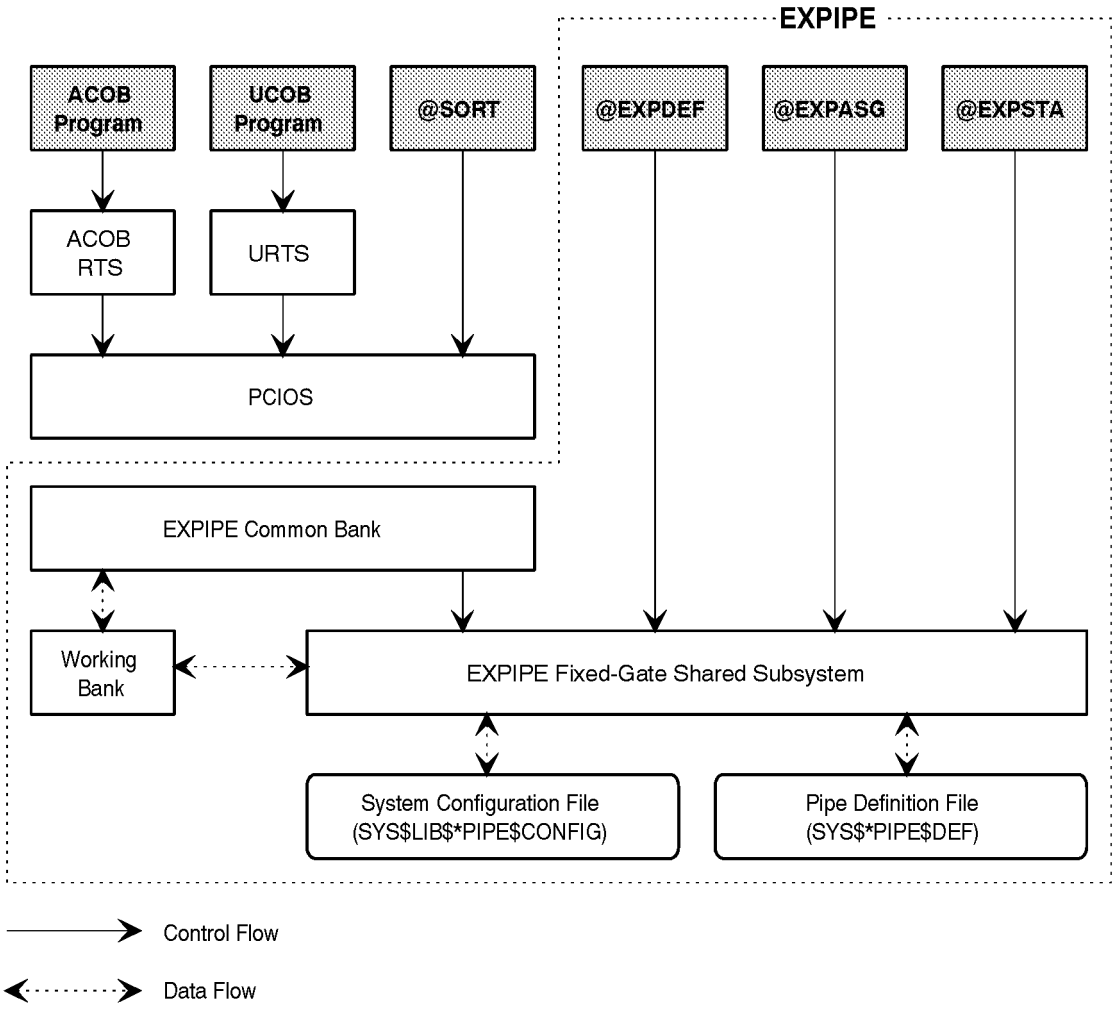


Figure 6–1. Structure of Expipe

Major characteristics of the internal structure of Expipe are as follows:

- The Expipe FGSS is a protected subsystem, which is the kernel of Expipe.
- The Expipe basic mode common bank supports mode switching between the basic mode PCIOS and the extended mode Expipe FGSS.
- The utility processors @EXPDEF, @EXPASG, and @EXPSTA are packaged as ZOOMs.
- The pipe definition file SYS$*PIPE$DEF is created as a catalogued file when Expipe is used. It contains information about pipe definitions.
- The system configuration file SYSLIB*PIPE$CONFIG is created as a catalogued file when Expipe is installed. It contains configuration information for the Expipe system. The system administrator can change the configuration parameter in this file if necessary.

Section 7
Expipe Installation and Configuration

This section describes how to install Expipe in your system. It also describes how to change Expipe system configuration parameters.

7.1. Installing Expipe

SOLAR should be used to install Expipe. Use the following procedure to install Expipe using SOLAR from the system console:

1. Start the INSTALLPKG runstream by the following ST keyin with the account-id, user-id, and password that have the privileges required for software installation:

   ```
   ST INSTALLPKG,,,account-id/user-id
   ```

2. Specify "tape" in response to the following prompt displayed by the INSTALLPKG runstream in the following manner:

   ```
   n–INSTALL FROM <TAPE>, DISK, OR RSS?
   n tape
   ```

3. Provide the reel numbers, assign options, and device type of the tapes in response to the prompt displayed by the INSTALLPKG runstream in the following manner:

   ```
   n–ENTER REEL NUMBER
   n reel-id[/reel-id...],options,type
   ```

   where:

   options

   is TJ for an unlabeled table, TF for a labeled tape (tape number checked), or T for a fully-labeled tape (tape name and number checked).

   type

   is U9S for a 6250 bpi open reel tape, or HICL or HIC40 for a U40 cartridge tape.

7.2. Installation Verification

The Expipe release tape contains a procedure that verifies whether Expipe has been properly installed and is ready to use. This installation verification procedure (IVP) consists of runstreams and programs to ensure that Expipe is installed and functional. The IVP is copied to the product file created by the installation process.

There are three IVP runstreams: IVP/ACOB, IVP/UCOB, and IVP/SORT.

You should note the following for these runstreams:

- ASCII COBOL is required for executing IVP/ACOB and IVP/SORT.
- UCS COBOL is required for executing IVP/UCOB.
- SORT is required for executing the IVP/SORT.

Use the following procedure to execute the Expipe IVP:

1. Enter one of the following ST keyins from the system console to start the Expipe IVP runstream named IVP/ACOB, IVP/UCOB, or IVP/SORT:

   ```
   ST SYS$LIB$*EXPIPE.IVP/ACOB,,,account-id/user-id  (for ACOB)

   ST SYS$LIB$*EXPIPE.IVP/UCOB,,,account-id/user-id (for UCOB)

   ST SYS$LIB$*EXPIPE.IVP/SORT,,,account-id/user-id (for SORT)
   ```

   Each of these runstreams automatically starts two additional IVP runstreams (IVP*x*1 and IVP*x*2).

2. All runs should finish within a few minutes. Expipe has been properly installed and is ready to use if all runs finish normally.

7.3. System Configuration – SYSLIB*PIPE$CONFIG

The system configuration file SYSLIB*PIPE$CONFIG is created as a cataloged file when Expipe is installed. This file is a symbolic system definition file (SSDEF) and contains the parameters ALLABORT and BUFNUM. These two parameters define the configuration for your Expipe system.

```
ALLABORT = NO
BUFNUM = 100
```

The system administrator can change the value of these parameters, if necessary, using the following procedure:

1. Install Expipe in your system, but do not start the IVP runstream.
2. Change a parameter value using a symbolic editor such as IPF.
3. Execute @EXPSTA control statement with option C, and confirm that the configuration value has been changed.
4. Start the LIBSAVE run if needed.

7.3.1. ALLABORT Parameter

Multiple reading programs can connect to a pipe. This parameter selects an action for the writing program if one of the reading programs has terminated in error.

Format

```
ALLABORT = { NO  }
           { YES }
```

Explanation

NO

As long as there is at least one remaining reading program executing normally, the writing program will not terminate in error. This is the default value.

YES

The writing program and all of the reading programs will immediately terminate in error if one of the reading programs has terminated in error.

Note: *One or more space characters are required before and after the equal sign (=).*

7.3.2. BUFNUM Parameter

This parameter specifies the maximum number of pipes concurrently used in a system.

Format

BUFNUM = *number*

Explanation

The *number* value also specifies the maximum number of FIFO buffer banks that have high BDIs (more than 010000) at APPLICATION level. If the Exec configuration parameter APLDYNBDS is not set high enough to allocate FIFO buffer banks, you must reduce the value of *number*. You can specify a value between 20 and 100. The default is 100.

Note: *One or more space characters are required before and after the equal sign (=).*

Figure 7–1 shows how FIFO buffer banks and data banks are allocated in an Expipe FGSS.

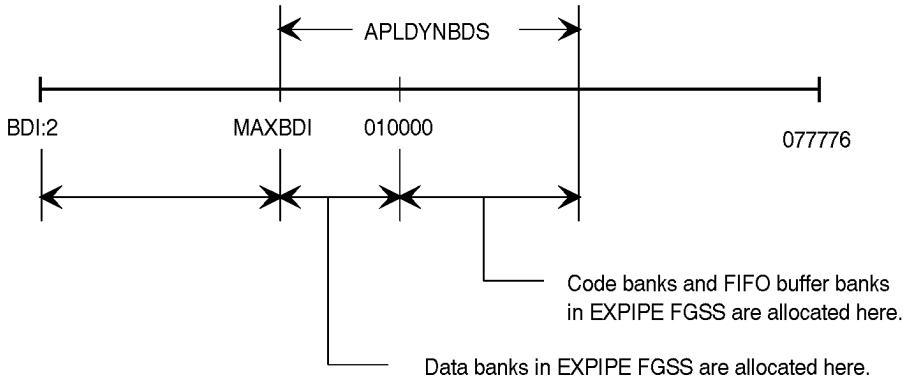


Figure 7–1. Allocation of BDIs at APPLICATION Level

Section 8
Diagnostic Messages

This section lists and describes diagnostic messages issued by Expipe and its utility processors.

8.1. Diagnostic Message Format on Print File

The following is the format of diagnostic messages displayed on print file:

```
EXPIPEnnn (message-class) : message text
```

where:

nnn

is a number consisting of three digits, with the first (most significant) digit having the following meaning:

Most Significant Digit	Meaning
1	Message was issued by one of the Expipe utility processors.
2	Message was issued by the utility portion of the Expipe fixed-gate shared subsystem (FGSS).
3	Message was issued by the kernel portion of the Expipe FGSS.

For internal or contingency errors, additional information may be displayed after the diagnostic messages.

8.2. Diagnostic Messages Issued by Expipe Utility Processors

`101: Completed`

The execution of a utility processor has completed successfully.

`102(ERROR): Illegal option specified`

At least one illegal option was specified on @EXPDEF, @EXPASG, or @EXPSTA processor call statement.

`103(ERROR): Pipe_name not specified`

No pipe_name is specified for @EXPDEF or @EXPASG processor call statement.

`104(ERROR): Illegal unit specified, BLOCK or KW is allowed`

An illegal value is specified as unit for @EXPDEF processor call statement. The legal values are BLOCK or KW only.

`105(ERROR): Write or read run_id not specified`

Write or read run-id is not specified for @EXPDEF processor call statement. Only one write run-id must be specified. Also at least one read run-id must be specified.

`106(ERROR): Too many read run_id, max is 3`

Four or more read runs are specified for @EXPDEF processor call statement. Up to three read run-ids can be specified.

`107(ERROR): Neither pipe_name nor option specified`

Neither pipe_name nor option is specified for @EXPSTA processor call statement.

`108(INTERNAL): ER CLIST$ fail (status=`*nnnnnnnnnnnn*`)`

ER CLIST$ has returned an error status of *nnnnnnnnnnnn* in the A0 register.

`109(INTERNAL): ER SYMB$ fail (status=`*nn*`)`

ER SYMB$ has returned an error status of *nn*.

`110(INTERNAL): The READ_INFOR error occurs(status=`*nnnnnnnnnnnn*`)`

The READ_INFOR routine has returned an error status of *nnnnnnnnnnnn*.

`111(INTERNAL): The SEARCH_INFOR error occurs (status=`*nnnnnnnnnnnn*`)`

The SEARCH_INFOR routine has returned an error status of *nnnnnnnnnnnn*.

`112(INTERNAL): S$CONV fail (status=`*nnnnnnnnnnnn*`)`

S$CONV has returned an error status of *nnnnnnnnnnnn*.

`113(INTERNAL): Cannot assign $$DMY (status=`*nnnnnnnnnnnn*`)`

Expipe failed to assign the internal working file named $$DMY. *nnnnnnnnnnnn* indicates the status code for @ASG control statement.

`115(INTERNAL): Cannot get qualifier of $$DMY`

Expipe failed to get qualifier of the internal working file named $$DMY.

`116(INTERNAL): Cannot free $$DMY (status=`*nnnnnnnnnnnn*`)`

Expipe failed to free the internal working file named $$DMY. *nnnnnnnnnnnn* indicates the status code for the @FREE control statement.

8.3. Diagnostic Messages Issued by Utility Portion of Expipe Fixed-Gate Shared Subsystem

201(ERROR): Write error on SYS$*PIPE$DEF (status=*nnnnnnnnnnnn*)

Expipe failed to write to the pipe definition file named SYS$*PIPE$DEF. *nnnnnnnnnnnn* indicates the I/O status code.

202(ERROR): Cannot assign SYS$*PIPE$DEF file (status=*nnnnnnnnnnnn*)

Expipe failed to assign the pipe definition file named SYS$*PIPE$DEF. *nnnnnnnnnnnn* indicates the status code for @ASG control statement.

203(ERROR): Read error on SYS$*PIPE$DEF (status=*nnnnnnnnnnnn*)

Expipe failed to read from the pipe definition file named SYS$*PIPE$DEF. *nnnnnnnnnnnn* indicates the I/O status code.

204(ERROR): “WAITBUFF” pipe is not found

No WAITBUFF pipe is found for @EXPSTA processor call statement with option F.

205(ERROR): “ACTIVE” pipe is not found

No ACTIVE pipe is found for @EXPSTA processor call statement with option A.

206(ERROR): “USED” pipe is not found

No USED pipe is found for @EXPSTA processor call statement with option U.

207(ERROR): “WAITING” pipe is not found

No WAITING pipe is found for @EXPSTA processor call statement with option W.

208(ERROR): “SLEEP” pipe is not found

No SLEEP pipe is found for @EXPSTA processor call statement with option S.

209(ERROR): “ACTIVE,” “USED,” “WAITING,” and “SLEEP” pipe are not found

No ACTIVE, SLEEP, USED, and WAITING pipe are found for @EXPSTA processor call statement with option L. That is, no pipe is currently defined.

210(ERROR): Illegal character in pipe_name

At least one illegal character is specified in pipe_name.

```
211(ERROR): Pipe-name already defined – replaced
```

The pipe name specified in the @EXPDEF processor call statement is already defined. The previous definition of the pipe is replaced by the new definition.

```
212(ERROR): Illegal character in output run_id
```

At least one illegal character is specified in output run-id.

```
213(ERROR): Number of read runs is invalid
```

The number of read runs is out of the allowed range of one to three.

```
214(ERROR): Illegal character in input run_id
```

At least one illegal character is specified in input run-id.

```
215(INTERNAL): S$CONV fail (status=nnnnnnnnnnnn)
Error element name : elt-name
```

The conversion by S$CONV failed. *nnnnnnnnnnnn* indicates the error status code.

```
216(ERROR): Cannot perform EXPASG for “ACTIVE” or “WAITBUFF” pipe
```

@EXPASG is not allowed for the ACTIVE or WAITBUFF pipe. A writing run is running or was erroneously terminated by an X keyin.

```
217(ERROR): Cannot perform EXPASG because pipeconnect function already performed
```

@EXPASG is not allowed because the specified pipe is in an ACTIVE or WAIT state. A reading run is running or was erroneously terminated by an X keyin.

```
218(ERROR): Cannot rerun this input run before the corresponding output
run does not terminate
```

The input run (reading run) cannot be executed again because the corresponding output run (writing run) is still running.

```
219(INTERNAL): The BDI value of FIFO bank is NULL
```

Expipe cannot get a BDI value assigned to a FIFO bank.

```
220(ERROR): Fifo_unit not correct
```

An illegal value is specified as unit for @EXPDEF processor call statement. The legal values are BLOCK or KW only. There may be an internal error within @EXPDEF processor.

```
221(ERROR): Buffer size is out of range. Should be 65KW through 262KW.
```

The value specified as the FIFO buffer size is out of the allowed range of 65KW through 262K words. The value less than 65K words is replaced by 65KW; whereas, the value larger than 262K words is replaced by 262KW.

```
230(ERROR): Pipe_name is not defined
```

No pipe name is specified for @EXPDEF processor call statement. There may be an internal error within @EXPDEF processor.

```
231(ERROR): Your run cannot assign this pipe
```

The @EXPASG assignment of a pipe to a run has failed because the run_id was not specified by @EXPDEF processor call statement.

```
232(ERROR): Too many pipes defined, max 2000
```

The @EXPDEF definition of a pipe has failed because 2000 pipes are already defined in the system.

```
233(ERROR): Cannot free this temporary file (status=nnnnnnnnnnnn)
```

@EXPASG has failed to free a temporary file having the same name as the pipe prior to assigning that file again. *nnnnnnnnnnnn* indicates the status code for the @FREE control statement.

```
234(INTERNAL): Cannot catalogue SYS$*PIPE$DEF (status=nnnnnnnnnnnn)
```

@CAT SYS$*PIPE$DEF returned an error status of *nnnnnnnnnnnn*.

```
235(INTERNAL): Cannot assign SYS$*PIPE$DEF at reboot or I-boot time
               (status=nnnnnnnnnnnn)
```

@ASG,A SYS$*PIPE$DEF returned an error status of *nnnnnnnnnnnn*. This message may be issued during the initial loading of an Expipe FGSS after recovery or an initial boot.

```
236(INTERNAL): Error occurs on UCSMAKEIOP$ at reboot time
               (status=nnnnnnnnnnnn)
```

An error was detected in a call to UCSMAKEIOPK$. Expipe cannot read the file SYS$*PIPE$DEF after a recovery boot.

```
238(INTERNAL): Free error on SYS$PIPE$DEF at reboot or I-boot time
               (status=nnnnnnnnnnnn)
```

@FREE SYS$*PIPE$DEF returned an error status of *nnnnnnnnnnnn*. This message may be issued during the initial loading of an Expipe FGSS after recovery or an initial boot.

```
239(INTERNAL): REG4TERM fail (status=nnnnnnnnnnnn)
```

The REG4TERM returned an error status of *nnnnnnnnnnnn*.

```
240(ERROR): This pipe-name is already defined and active (status=nnnnnnnnnnnn)
```

@EXPDEF has failed because the pipe named on the @EXPDEF processor call is already defined and active (in use). This @EXPDEF call is ignored.

```
241(ERROR): The output run id and input run id are duplicated
```

@EXPDEF has failed because the same run-id is specified for both the output and input runs. Different run-ids must be specified for output and input runs; none of them can be the same.

```
242(INTERNAL): Free error on SYS$*PIPE$DEF (status=nnnnnnnnnnnn)
```

Expipe has failed to free the pipe definition file named SYS$*PIPE$DEF. *nnnnnnnnnnnn* indicates the status code for the @FREE control statement.

```
243(ERROR): Cannot assign this temporary file (status=nnnnnnnnnnnn)
```

@EXPASG has failed to assign a temporary file having the same name as the pipe. *nnnnnnnnnnnn* indicates the status code for the @ASG control statement.

```
244(ERROR): Write error on temporary file (status=nnnnnnnnnnnn)
```

@EXPASG has failed to write to a temporary file having the same name as the pipe. *nnnnnnnnnnnn* indicates the I/O status code.

```
245(INTERNAL): Cannot use attach internal file name (@USE) to temporary file
```

Expipe has failed to attach an @USE internal file name to a temporary file having the same name as the pipe.

```
246(INTERNAL): Cannot free $$TEMP file with A option
```

Expipe has failed to free (@FREE,A) the internal file name attached to a temporary file having the same name as the pipe.

```
247(INTERNAL): Cannot display console message (status=nnnnnnnnnnnn)
Element name : elt-name
```

Expipe has failed to display a message upon console. *nnnnnnnnnnnn* indicates the error status code.

```
248(INTERNAL): LS$BANK_LIST fail (status=nnnnnnnnnnnn)
```

The Linking System's program callable interface LS$BANK_LIST returned an error status of *nnnnnnnnnnnn* when Expipe called LS$BANK_LIST to obtain BDI values of banks.

```
249(INTERNAL):S_STORAGE_INSPECT fail(status=nnnnnnnnnnnn)
Bank name : bank-name  Bank BDI : BDI-value
```

A call to S_STORAGE_INSPECT has failed. *nnnnnnnnnnnn* indicates the error status code.

```
250(ERROR): The definition cannot delete an active or waiting pipe
```

An attempt to delete a pipe (using an @EXPDEF processor call statement with option D) has failed because the status of the pipe is active (in use) or waiting. The @EXPDEF call is ignored.

```
251(INTERNAL): Error occurs on UCSMAKEIOPK$
```

An error is detected in a call to UCSMAKEIOPK$. Expipe calls UCSMAKEIOPK$ to make an I/O packet to write to the pipe definition file SYS$*PIPE$DEF.

```
260(ERROR): Cannot assign file SYS$LIB$*PIPE$CONFIG; default value is assumed
```

The pipe configuration file SYSLIB*CONFIG cannot be assigned. Default values are used as BUFNUM and ALLABORT parameters.

```
262(ERROR): SYS$LIB$*PIPE$CONFIG is exclusively assigned by another run;
default value is assumed
```

The pipe configuration file SYSLIB*CONFIG is exclusively assigned by another run. Default values are used as BUFNUM and ALLABORT parameters.

```
263(ERROR): Read error on file SYS$LIB$*PIPE$CONFIG; default value is assumed
```

Expipe cannot read the configuration file SYSLIB*CONFIG. Default values are used as BUFNUM and ALLABORT parameters.

```
265(ERROR): Specified parameters in file SYS$LIB$*PIPE$CONFIG are illegal;
default value orlatest correct value is assumed
```

A syntax error was detected in the configuration file `SYS$LIB$*PIPE$CONFIG`. Default values (or the latest correct values) are assumed for the BUFNUM and ALLABORT parameters.

```
266(ERROR): Cannot free file SYS$LIB$*PIPE$CONFIG
```

The pipe configuration file SYSLIB*CONFIG cannot be freed.

```
270(ERROR): Pipe initialization in progress; Do not enter @@X or @@X T
```

At initial loading of the Expipe subsystem, the keyins @@X, @@X T, and @@X E are not allowed. This warning message will be displayed from @EXPDEF, @EXPASG, or @EXPSTA only for the first demand user following a system boot.

8.4. Diagnostic Messages Issued by Kernel Portion of Expipe Fixed-Gate Shared Subsystem

```
300(ERROR): Cannot allocate FIFO buffer; replace 262K word
```

The size of FIFO buffer (specified by *n*/BLOCK on the @EXPDEF call) exceeds 262K words. Expipe allocates a FIFO buffer size of 262K words when creating the pipe.

```
301(ERROR): Cannot perform pipecreate function for “ACTIVE” or “WAIT/BANK”
pipe – qual*file
```

The pipe *qual*file* cannot be created because it is already active (in use). Possible causes include the following:

- Two or more writing runs are executing.
- An X keyin was issued to a run using the pipe, making the pipe purge incorrectly.

```
302(INTERNAL): The condition of pipe is not ACTIVE, WAIT, SLEEP or USED
```

The status of the pipe is in error; it is not active, waiting, sleep, or used.

```
303(ERROR): Cannot continue piperead process because the output run has
terminated in error mode
```

This reading run cannot continue the read access to the pipe because the writing run has terminated in error. It is necessary to inspect the status of the writing run.

```
304(ERROR): Not sequential I/O
```

The data block specified by a read or write request is out of sequence.

```
305(ERROR): This function is invalid
Error element name : elt-name
```

An invalid function has been requested to Expipe.

```
306(ERROR): The value of word count is not a multiple of 28
```

The value of word count for a read or write request to Expipe is not a multiple of 28.

```
308(ERROR): Cannot perform pipeconnect function because pipeconnect
function already performed
External pipe name is qual*file
```

The pipe *qual*file* cannot be connected to this reading run because the pipe is already active (in use) or waiting and connected. Possible causes include the following:

- Two or more reading runs with the same original run-id are executing.
- An X keyin was issued to a run using a pipe, making the pipe purge incorrectly.

```
309(INTERNAL): Error occurs in this output run
Output runid is run-id
External pipe name is qual*file
```

An error has occurred in this writing run. This message is issued by the termination routine of Expipe.

```
310(INTERNAL): Error occurs in this input run
Input runid is run-id
External pipe name is qual*file
```

An error has occurred in this reading run. This message is issued by the termination routine of Expipe.

```
311(INTERNAL): This pipe is locked
The number of PCE is number
```

The lock cell is locked for this pipe. This message is issued by the termination routine of Expipe.

```
312(INTERNAL): This bank is not FIFO bank
Bank BDI is BDI-value
```

This bank is not used for FIFO buffer for a pipe.

```
313(ERROR): The sector address of PIPEPREREAD must be 0
```

The sector address was not zero when the SORT processor made the first read request to a pipe. It must be zero.

```
314(ERROR): The word count of PIPEPREREAD must be 28
```

The word count is not 28 when the SORT processor made the first read request to a pipe. It must be 28.

```
315(ERROR): This external pipe name is not defined – qual*file
```

The external pipe *qual*file* is not defined by @EXPDEF.

```
316(INTERNAL): This external pipe name must be defined in internal table
after pipepreread – qual*file
```

The external pipe *qual*file* is not defined in an internal table of Expipe, where it should have been defined as a result of the first read request to the pipe by the SORT processor.

```
317(INTERNAL): Read request word count is larger than FIFO buffer bank
```

An illegal read request was made that specifies a word count larger than the size of FIFO buffer bank.

```
318(INTERNAL): Cannot allocate FIFO buffer (status=nnnnnnnnnnnn)
```

The Exec extended mode call MODIFY$BANK has returned error status *nnnnnnnnnnnn* when Expipe called MODIFY$BANK to allocate a FIFO buffer bank for a pipe.

```
319(INTERNAL): Cannot free FIFO buffer (status=nnnnnnnnnnnn)
```

The Exec extended mode call MODIFY$BANK has returned error status *nnnnnnnnnnnn* when Expipe called MODIFY$BANK to release a FIFO buffer bank for a pipe.

```
320(ERROR): Your run cannot use this pipe
```

This run cannot use this pipe because the run-id has not been specified on @EXPDEF processor call statement.

```
321(INTERNAL): Mass storage-addr is max logical page over
```

```
322(ERROR): Cannot perform pipepreread function because pipeconnect
function already performed – qual*file
```

The SORT processor has made its first read request to the pipe *qual*file* but failed because the pipe is already connected. Possible causes include the following:

- Two or more reading runs with the same original run-id are executing.
- An X keyin was issued to a run using a pipe, making the pipe purge incorrectly.

```
323(INTERNAL): Console message cannot be issued (status=nnnnnnnnnnnn)
```

Expipe could not issue a console message.

```
328(INTERNAL): LS$BANK_NAME fail (status=nnnnnnnnnnnn)
Bank BDI is BDI-value
```

The Linking System's program callable interface LS$BANK_NAME has returned error status *nnnnnnnnnnnn* when Expipe called LS$BANK_NAME to obtain bank names.

```
329(INTERNAL): LS$VA_OM_REL fail (status=nnnnnnnnnnnn)
Bank BDI is BDI-value
```

The Linking System's program callable interface LS$VA_OM_REL has returned an error status *nnnnnnnnnnnn*, when Expipe calls LS$VA_OM_REL to obtain a relative address (in object module) for a virtual address.

```
330(CONTINGENCY): Contingency error occurs in Expipe
```

A contingency error has occurred in the Expipe fixed-gate shared subsystem (FGSS).

```
331(CONTINGENCY): Contingency error occurs in contingency processing
```

A contingency error has occurred during contingency processing of Expipe, making it impossible for Expipe to continue contingency processing.

```
331(CONTINGENCY): Expipe contingency processing terminated
```

Expipe cannot continue contingency processing and has terminated.

```
333(INTERNAL): REG4TERM fail (status=nnnnnnnnnnnn)
Element name is elt-name
```

The Exec extended mode call REG4TERM has returned error status *nnnnnnnnnnnn* when Expipe called REG4TERM to register an activity termination routine.

```
334(INTERNAL): S_STORAGE_INSPECT fail (status=nnnnnnnnnnnn)
```

A call to S_STORAGE_INSPECT has failed. *nnnnnnnnnnnn* indicates the error status code.

```
335(INTERNAL): ER BANK$ cannot produce a PROGRAM-level working storage bank
```

ER BANK$ has failed when Expipe calls ER BANK$ to acquire a program level working storage bank.

```
336(INTERNAL):The upper limit of PROGRAM-level working storage bank is not
031777
```

The program level working storage bank has failed to meet the Expipe requirement that its upper limit must be 031777.

```
337(INTERNAL):The lower limit of PROGRAM-level working storage bank is not
030000
```

The program level working storage bank has failed to meet the Expipe requirement that its lower limit must be 030000.

```
338(INTERNAL):The call of RTS$NPESTAT fail
```

The URTS TPE-to-NPE utility routine RTS$NPESTAT has returned an error status (a status other than 0 or 1) when the Expipe common bank calls RTS$NPESTAT to see if the NPE environment already exists.

```
339(INTERNAL):The call of RTS$TPENPE fail
```

The URTS TPE-to-NPE utility routine RTS$TPENPE has returned an error status when the Expipe common bank calls RTS$TPENPE to call the Expipe fixed-gate shared subsystem (FGSS).

```
340(INTERNAL):The call of RTS$NPESETUP fail
```

The URTS TPE-to-NPE utility routine RTS$NPESETUP has returned an error status when the Expipe common bank calls RTS$NPESETUP to call the Expipe fixed–gate shared subsystem (FGSS).

```
341(INTERNAL):The locality of this bank is not PROGRAM_LOCAL
Bank name : name  Bank locality : locality
```

The locality of this bank is not PROGRAM local, which is required by Expipe.

```
342(INTERNAL):The locality of this bank is not ACTIVITY_LOCAL
Bank name : name  Bank locality : locality
```

The locality of this bank is not ACTIVITY local, which is required by Expipe.

```
343(INTERNAL):The locality of this bank is not APPLICATION_LOCAL
Bank name : name  Bank locality : locality
```

The locality of this bank is not APPLICATION local, which is required by Expipe.

```
344(INTERNAL):This program bank does not have the header('PIPE')
```

This program bank does not have the identifier (ASCII character string 'PIPE') which indicates the bank is used by Expipe.

```
345(CONTINGENCY): Contingency routine terminated successfully
```

Expipe contingency routine has terminated its processing successfully.

```
346(INTERNAL): External pipe name corresponding to internal pipe name was not
found – pipe–name
```

Expipe has failed to find the external pipe name corresponding to the internal pipe named *pipe–name*.

```
347(ERROR): The number of pipe exceeds max number(20) in a program – pipe–name
External pipe name is qual*file
```

This program cannot use the pipe named *pipe–name* because it already uses 20 pipes. Expipe allows a program (or a run) to use up to 20 pipes concurrently.

348(ERROR): External pipe name cannot find internal file table – *qual*file* in *qual*file*

Expipe has failed to find the external pipe *qual*file* in its internal table.

349(ERROR): Cannot continue pipewrite process because all input (reading) runs have terminated in error mode (ALLABORT = NO)

This writing program cannot continue to write to a pipe, because all reading runs for the pipe have terminated in error. It is necessary to check the status of the reading runs.

350(ERROR): Cannot rerun this input run until the corresponding output run terminates

This reading run cannot be rerun because the corresponding writing run is still running although this reading run has terminated prematurely.

351(ERROR): The value of word_count exceeds 65k words

The value of word count for read or write request to Expipe exceeds 65,535.

353(ERROR): The value of word_count is 0 word

The value of word count for read or write request to Expipe is zero.

354(INTERNAL): Internal Pipe name cannot be found in internal file table - *qual*file*

An internal pipe name cannot be found in the pipe definition table.

355(ERROR): Cannot continue pipewrite process because input run has terminated in error mode (ALLABORT = YES)

The output run (writing run) cannot continue to write data into a pipe because one of the input runs (reading runs) terminated in error mode. This message is issued when the configuration parameter ALLABORT = YES.

400(INTERNAL): Clear bdi_lock

Expipe (EXPCLR) unlocked the BDI table.

401(INTERNAL): Clear pce_initial_lock

Expipe (EXPCLR) unlocked the initial_stat_flag of the PCE table.

402(INTERNAL): Clear tbl_lock

Expipe (EXPCLR) unlocked the PCE table.

403(INTERNAL): Clear bdi_lock and pce_initial_lock

Expipe (EXPCLR) unlocked the BDI table and the initial_stat_flag of the PCE table.

```
404(INTERNAL): Clear bdi_lock and tbl_lock
```

Expipe (EXPCLR) unlocked the BDI table and PCE table.

```
405(INTERNAL): Clear pce_initial_lock and tbl_lock
```

Expipe (EXPCLR) unlocked the initial_stat_flag of the PCE table, and the PCE table itself.

```
406(INTERNAL): Clear bdi_lock, pce_initial_lock, and tbl_lock
```

Expipe (EXPCLR) unlocked the BDI table and the initial_stat_flag of the PCE table, and the PCE table itself.

8.5. Console Messages

*run-id**EXPIPE701: *qual*file* HOLD FOR *nnn* MINS, *run_o*

Meaning

The reading run (*run-id*) is waiting *nnn* minutes for the pipe *qual*file* to be created by the writing run (*run_o*).

Action

Inspect the status of the writing run (run_o).

*run-id**EXPIPE702: *qual*file* FULL FOR *nnn* MINS, *run_i*

Meaning

The writing run (*run-id*) is waiting *nnn* minutes because the pipe *qual*file* has been full, and the reading run (*run_i*) does not read data blocks from the pipe.

Action

Inspect the status of the reading run *run_i*.

*run-id**EXPIPE703: *qual*file* EMPTY FOR *nnn* MINS, *run_o*

Meaning

The reading run (*run-id*) is waiting *nnn* minutes because the pipe *qual*file* has been empty and the writing run (*run_o*) does not write data blocks to the pipe.

Action

Inspect the status of the writing run (run_o).

*run-id**EXPIPE704: *qual*file* ASSIGN FOR *nnn* MINS, *run_o*

Meaning

The reading run (*run-id*) is waiting *nnn* minutes for the pipe *qual*file* to be assigned by the writing run (*run_o*).

Action

Inspect the status of the writing run (*run_o*).

*run-id**EXPIPE705: *qual*file* HOLD FOR *nnn* MINS,*run_i1*[,*run_i2*[,*run_i3*]]

Meaning

The writing run (run-id) is waiting *nnn* minutes for the reading run(s), such as *run_i1*, to terminate the execution of the program(s) reading from the pipe *qual*file*. The run-ids indicated are the reading runs, which are executing the programs that are reading from the pipe.

Action

Inspect the status of the reading runs (*run_i1*, *run_i2*, and *run_i3*).

*run-id**EXPIPE706: *qual*file* HOLD FOR *nnn* MINS

Meaning

The writing run (run-id) is waiting *nnn* minutes because 100 pipes are in use, and no FIFO buffer is available for the pipe *qual*file*.

Action

Inspect the status of the other runs using Expipe.

8.6. Other Messages

PCIOS returns the following error codes and messages to its caller when an error other than a contingency has taken place in Expipe. ASCII COBOL, UCS COBOL, and the SORT processor issue their own message text as specified in the following table:

Error Code	Error Message and Explanation
1	An error occurred while using a pipe. A diagnostic message by Expipe precedes this message. For UCS COBOL: *ERROR(URTS) 35001:FILE SYSTEM - 1 I/O error returned - Code: 000011 For ASCII COBOL and SORT: I/O STATUS CODE 11
31	READ/WRITE TYPE INHIBITED FOR THIS FILE Specified type of read or write operation is not allowed for the pipe. Possible causes include: • Read/write requests by programs other than ACOB, UCOB, and SORT • Use of features not supported by Expipe, such as COBOL OPEN EXTEND statement For UCS COBOL: *ERROR(URTS) 19002:FILE SYSTEM - 90 Expipe is not allowed For ASCII COBOL and SORT: Expipe IS NOT ALLOWED

8.7. Dump List for Expipe Internal Errors

When an unrecoverable contingency error has taken place in Expipe FGSS, Expipe issues an error message and prints a dump list, which is used for problem analysis.

The dump list has the following characteristics:

- File name Expipe*CGYDUMP.
- Run–id of the the run using Expipe
- Standard print queue named PR

8.8. Reporting Problems

The following materials are required for prompt resolution of reported problems that pertain to Expipe:

- Program execution listing that shows the problem
- Listing created by @EXPSTA,L processor call statement
- Dump listing produced by Expipe with the run–id identifying the program using it (when a contingency takes place in the Expipe FGSS)

Bibliography

OS 1100 ASCII COBOL Programming Reference Manual (UP-8582). Unisys Corporation.

OS 2200 Sort/Merge Programming Guide (7831 0687). Unisys Corporation.

OS 2200 Universal Compiling System (UCS) COBOL Programming Reference Manual Volume 1, COBOL Statements (7831 0448). Unisys Corporation.

OS 2200 Universal Compiling System (UCS) COBOL Programming Reference Manual Volume 2, Compiler and System Interface (7831 0455). Unisys Corporation.

Index

Symbols

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

U

W

X